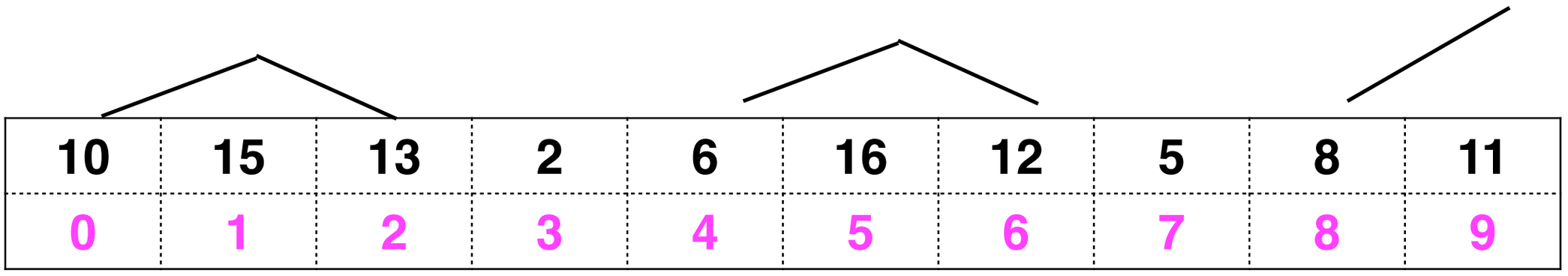
Il problema del massimo locale in un array

Dato un array di numeri interi, vogliamo individuare un massimo locale, cioè un elemento maggiore dei suoi due vicini, o dell'unico vicino presente se l'elemento è il primo o l'ultimo, in O(lg n).

Esempio:



10	15	13	2	6	16	12	5	8	11
0	1	2	3	4	5	6	7	8	9

Qui i massimi locali sono

15,
16 e
11

Soluzione lenta

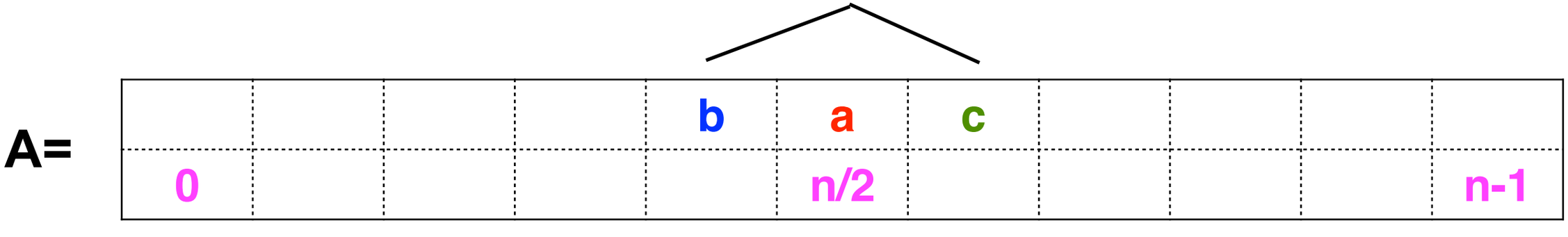
Si potrebbe scorrere l'array da sinistra a destra controllando la condizione di massimo locale. Quanti passi al più costa questo modo di procedere?

Costa <u>al più</u> lo scorrimento di tutto l'array, nel caso sia ordinato in ordine crescente, quando cioè l'unico massimo locale è il massimo dell'array.

Quindi θ(n) nel caso peggiore, θ(1) in quello migliore quindi O(n) in tutti i casi.

Dobbiamo risolverlo in O(lg n)!

E’ un problema di ricerca



Confrontiamo l’elemento centrale a con i suoi vicini:
se a ≥ b e a ≥ c allora a è un massimo locale.

Vediamo se sapendo che a < b o a < c possiamo restringere la ricerca a metà array.

Caso a < b: a destra?

A=

				b	a	c				
0					n/2					n-1

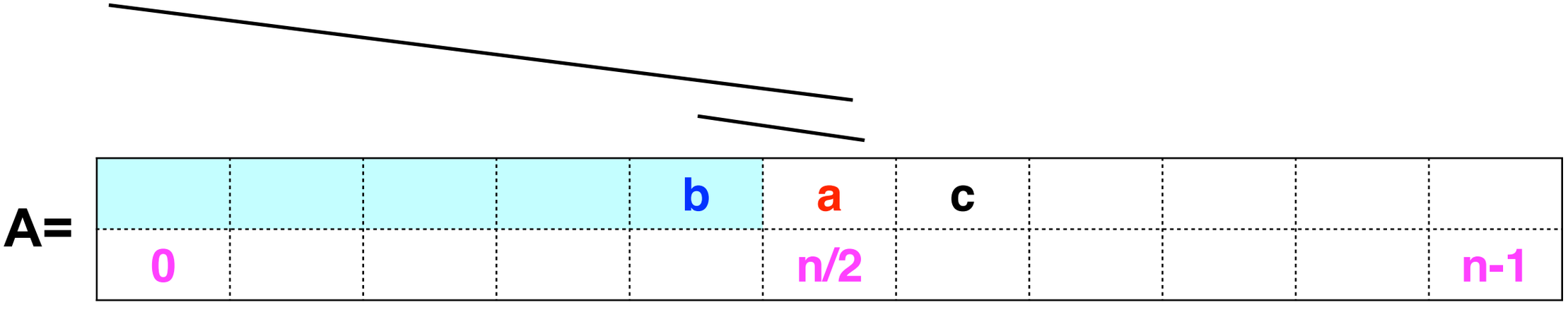
se a < b e andassi a destra?

A=

				b	a	c				
0					n/2					n-1

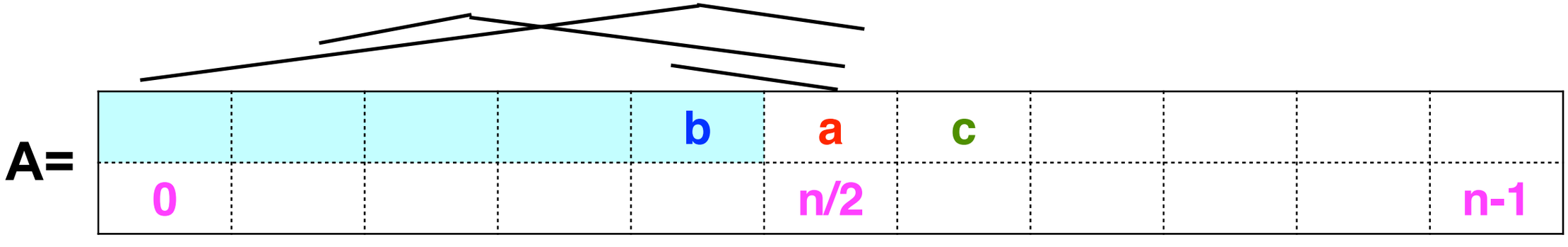
Non va, nella metà destra potrebbero essere in ordine decrescente e allora l'unico massimo locale nella metà destra sarebbe c, un massimo locale "al bordo" della porzione di array esaminata e che non è un massimo locale per tutto l'array se a ≥ c!

Caso a < b: a sinistra?



Se a < b possono verificarsi due casi:
Caso 1. tutti gli elementi di indice 0 fino a n/2 sono ordinati in ordine decrescente.
In tal caso un massimo locale è A[0], che è anche un massimo locale per tutto l'array.

Caso a < b: a sinistra?

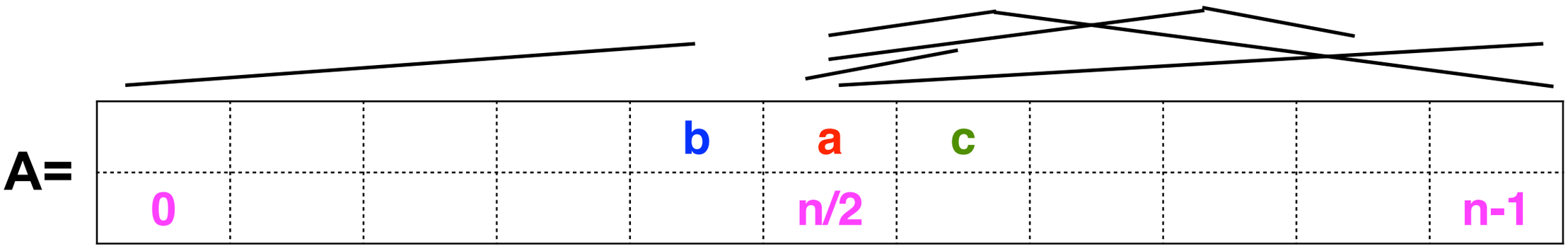


Se a < b possono verificarsi due casi:

Caso 2. Non tutti gli elementi di indice 0 fino a n/2 sono ordinati in ordine decrescente.
In tal caso c'è un cambio di tendenza e quindi un massimo locale.
Il cambio di tendenza potrebbe avvenire già con b, che quindi sarebbe un massimo locale al "bordo" per la metà sinistra dell'array, ma anche per tutto l'array, in virtù della disuguaglianza a < b.

Quindi in entrambi i casi nella prima metà dell'array c'è un massimo locale che è anche un massimo locale per tutto l'array.

Caso a < c: a destra o a sinistra?



Se a < c i casi sono analoghi a quanto visto per la metà sinistra..

A sinistra potrebbero essere tutti crescenti e quindi un massimo locale sarebbe b per la metà, ma non per tutto l'array se b < a.
Nella metà destra dell'array c'è un massimo locale che è anche massimo locale per tutto l'array, l'ultimo elemento se sono tutti crescenti alla destra di a,
un elemento intermedio se c'è un cambio nella crescita,
anche c stesso.

L'idea algoritmica per trovare un massimo locale

ripeti

If A[n/2] < A[n/2 − 1] dirigi la ricerca nella metà sinistra 0 . . . n/2 − 1
else if A[n/2] < A[n/2 + 1] dirigi la ricerca nella metà destra n/2 + 1 . . . n − 1
finché A[n/2] non è un massimo locale (cioè A[n/2] ≥ A[n/2 − 1] e A[n/2] ≥ A[n/2 + 1])

Se l'elemento centrale non è un massimo locale l'array nel quale si va a cercare si dimezza a ogni passo:

2	6	13	10	15	16	18	5	8	7	26
0	1	2	3	4	5	6	7	8	9	10

18	5	8	7	26
6	7	8	9	10

A[8] = 8 è un massimo locale.

Al più possiamo dimezzare lo spazio della ricerca fino ad arrivare ad 1, cioè $\log_2 n$ volte!
Osserviamo che la ricerca si dirige verso la metà dell'array in cui c'è sicuramente un massimo locale, quindi non c'è uscita dal ciclo per insuccesso.

Pseudocodice

```
MaxLoc(A)
INPUT: una sequenza A di interi.
prec: gli elementi di A sono tutti distinti e len(A) ≥ 2
OUTPUT: restituisce  la posizione di un massimo locale in A
lo = 0 e hi = len(A)  inizialmente il massimo locale può essere ovunque nel vettore
 nel ciclo che segue il picco è tra gli elementi A[lo],…,A[hi-1]
while lo < hi-1  do
     {m = (lo+hi)/2
         if  (m = hi-1) and (A[m] > A[m -1])) then return m  else return m-1
         if  (m = lo) and (A[m] > A[m +1]))    then return m else return m+1
          if  (A[m] > A[m − 1] and A[m] > A[m + 1]) then return m  A[m] è un massimo
        locale e tra lo e hi-1 ci sono almeno 3 elementi
          if (A[m] < A[m - 1])) ci sono almeno due elementi
                then hi = m   nella metà sinistra
                else
          if (A[m] < A[m + 1])) ci sono almeno due elementi
                then lo = m+1  nella metà destra
       }
return lo qui si esce se lo=hi-1 e l'elemento rimasto è un massimo locale, infatti
se hi=m, allora m=lo+1 e A[lo+1] < A[lo] allora A[lo] è un massimo locale. Oppure
se lo = m+1 allora m=hi-2 e A[m] < A[m+1] allora A[lo] è un massimo locale.
```

Lo pseudocodice con meno confronti

```
MaxLoc(A)
INPUT: una sequenza A di interi.
prec: len(A) ≥ 2
OUTPUT: restituisce  la posizione di un massimo locale in A
lo = 0 e hi = len(A) inizialmente il massimo locale può essere ovunque nel vettore
nel ciclo che segue il picco è tra gli elementi A[lo],…,A[hi-1]
while (lo < hi-1) do finché ci sono almeno due elementi
     m = (lo+hi)/2
     if  (m = hi-1) and (A[m] > A[m -1]))    then return m
     if  (m = lo) and (A[m] > A[m +1]))    then return m
     if (A[m] < A[m − 1]) then hi = m nella metà sinistra
     if (A[m] < A[m + 1]) then lo = m + 1 nella metà destra
     else return m A[m] è un massimo locale, perchè
     se m>lo e m <hi-1 e A[m] ≥ A[m-1] e A [m] ≥ A [m+1] allora m è  massimo locale
     “centrale” nella porzione di array che si considera
return lo  lo è un massimo locale, si vedano le spiegazioni nella versione precedente
```